



LINUX Expert

Share and share alike is the motto this month as **Jon Thompson** shows you how to build your own router so you can share your internet connection with a network of PCs – and that's just the start



■ SUSE LINUX PROFESSIONAL 9.3 RELEASED

Novell has released SuSE Linux Professional 9.3. Designed for home and desktop use, this has a host of features including the 2.6.11 kernel and the latest versions of the usual applications, such as OpenOffice 2.0.

Liphone is new to the package, providing a Voice over IP (VoIP) client. Mozilla's Firefox is also bundled. This is the default browser when using the Gnome desktop and is available as an option with the KDE desktop.

Other features include improved wireless networking, Bluetooth support and PDA and mobile phone synchronisation. More details are available at www.novell.com, and a review will appear in next month's *Linux Expert*.

■ DEBIAN FINDS NEW LEADER

The Debian project has a new leader. Brandon Robinson, a senior employee of Progeny Linux Systems, which has close ties to the Linux project, was finally elected after losing in a previous attempt. Robinson faces a number of challenges, not least of which is to establish a predictable release structure. The current development, codenamed Sarge, was due to arrive in late 2004 but is still unavailable.

Also on the agenda is Debian's not-for-profit funding structure. In Robinson's first report, he claims that the project has just over \$40,000 (around £21,000) of working capital. Also, many donors insist their money be used in short-term projects.

On a positive note, Debian's Asian exposure is increasing thanks in part to new partnerships with China's Sun Wah Linux and Japan's VA Linux Systems, both of which will help with Debian's development.

■ NEW KERNEL PROMISES A LOT

Sources close to the next Linux kernel say it promises great things. Version 2.6.12 will support both Trusted Computing and Security Enhanced Linux, along with Red Hat's address space randomisation idea, which is intended to prevent buffer overflow attacks. It will also feature virtualisation, which enables one computer to run several Linux systems, as well as enhanced clustering capabilities.

Building your own router using Linux may seem a little pointless when inexpensive ADSL modems already come with a router and firewall built in. However, low-cost consumer routers often lack serious features that businesses and Linux enthusiasts are after. If you buy a router that does everything you want, you face a bill of hundreds of pounds. Create your own router, though, and you'll be able to share your internet connection with a network of PCs, organise a group of networks securely and view detailed information about your network traffic. You may even breath new life into an old PC that was destined for the scrap heap.

Routers range from small, domestic units to ISP-class, rack-mounted devices with dozens of ports that can handle many gigabytes of information per second. Routers send data between different networks, enabling computers on one network to communicate with other computers on networks in the next room, city or country. Routers are the workhorses of the internet and internal networks alike.

Although routers make up the structure of the internet, no single router has an overview of every computer in the world. It knows only about its own networks, and everything else is a mystery to it. Without knowing the ultimate destination of your data, routers edge it ever closer by passing it to a neighbour that's nearer.

When data leaves your computer for a destination on the internet, it travels across your local network and arrives at the first router. If the destination address of this data packet is not available on any network connected to one of the router's network interfaces, the router sends it to another router that it has been told is closer to the internet. It uses a table to decide which interface it will use to re-transmit the packets. The router that receives this re-transmitted data checks the destination address and uses its own routing table to work out which interface it should use to send the data onwards. Each stage in the journey is known as a hop.

The data hops through all available routers until it reaches the internet, where it continues to move between ISPs' routers until it reaches its destination network. You can trace the route your data takes from your computer to a destination server using the traceroute command:

```
# traceroute www.linux.org#
```

The output of this command will look something like this:

```
traceroute to www.linux.org
(198.182.196.56), 30 hops max, 40 byte
packets
```

```
1 192.168.0.1 0.431 ms 0.514 ms
0.983 ms
2 * ge0-1.lns2-c4.brm.planet.net.uk
(62.25.204.137) 21.674 ms
24.127 ms
3 ge5-0.pbr-1.brm.as5388.net
(62.25.204.130) 25.070 ms 24.942 ms
25.900 ms
[lines removed for clarity]
21 router.invlogic.com
(207.245.34.122) 120.730 ms * *
22 www.linux.org (198.182.196.56)
126.470 ms 128.411 ms 129.389
ms
```

Don't be surprised if many destinations end in a series of asterisks. Some networks filter out traceroute traffic for security reasons.

Building a router from a computer running Linux and a couple of network interface cards (NICs) is straightforward and can save you money. You can also use it to serve DHCP addresses and secure it with its own firewall at the same time. What's more, you won't have to learn an arcane router command language.

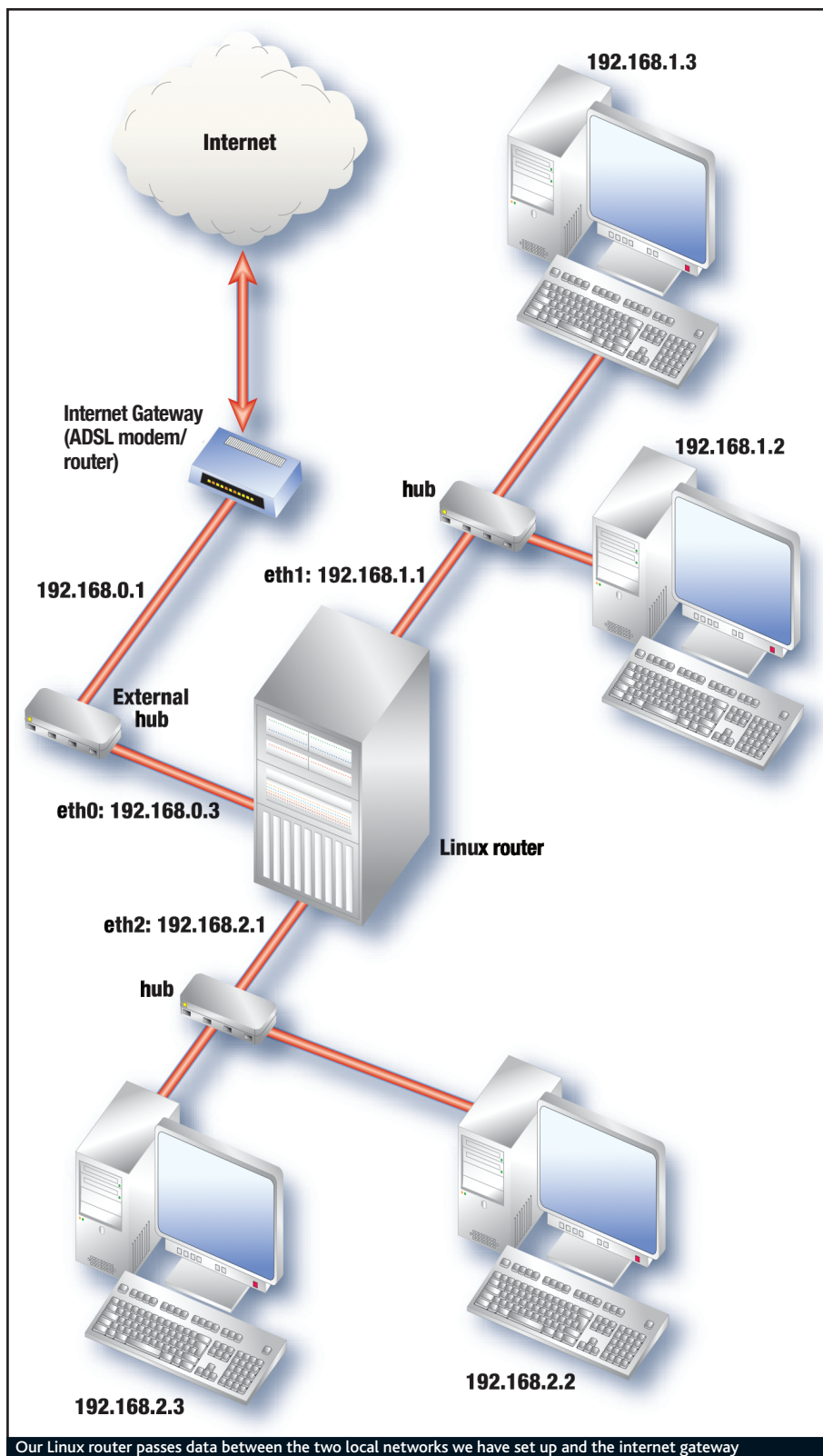
LOCAL ROUTES FOR LOCAL PACKETS

Here's an example of what happens to a packet of information when it's sent from one internal network to another. The system sending the data has an IP address of 192.168.1.2, and it wants to send the data to a server with an address of 192.168.2.2. These two computers are not on the same network – if they were, they would both have addresses of 192.168.1.x, where x is any number from one to 254. In this example, the first system is on a network called 192.168.1.x and has a machine number of 2. The server is on a network called 192.168.2.x and also has a machine number of 2.

Imagine that a router connects the two networks directly. When the first computer sends its data, it recognises that the destination server is not on its own network, so it sends the data to the router instead. But how does the router know where to send these data packets?

When you send a packet to a computer on another network, the router reads information embedded in the packet to extract the destination address. It then consults its routing table to see which interface it should use to re-transmit the packet. A routing table associates networks with the router's network interfaces.

Sometimes the destination network is one to which the router is not connected, such as 198.182.196.56 (www.linux.org), for example. In this case the router uses the default root value in its routing table. This gives the address of another router that may know what to do with our mystery destination address. In the case of our small



network, the router will send any packets with destinations that are not on the local network to the internet gateway. The gateway will then send it to the ISP for it to deal with.

The worldwide mesh of interconnected routers that forms the backbone of the internet is vast so, unless the destination computer is not running, your packets will usually get through. The underlying Transmission Control Protocol (TCP) makes sure that every packet carries a unique serial number, so if any get lost or arrive

out of sequence due to routing problems the communicating systems will realise and can reassemble or resend the data.

BUILDING THE ROUTER

As you can see in the diagram above, our router is plugged into a network with an existing ADSL modem/router, as supplied and configured by our ISP. This acts as the internet gateway. Some home networks still use old Windows 98 computers with a firewall and connection sharing as their internet

gateway. Our router doesn't care what you use, as long as the gateway forwards data successfully to and from the internet.

Our router connects to the internet gateway using its first NIC, and each of the remaining NICs serves an internal network. Some people like to connect their internet gateway device to the router using two normal cables with a hub between them, as we've done in our diagram. Alternatively, you can simplify matters by dispensing with the hub. You can't rely on a regular network cable to connect a computer to a router like this, you'll generally need to use a crossover cable.

Our router will run a clean installation of SuSE Linux 9.2 Professional, which we'll update using the YaST utility to load the latest system patches. When building a Linux router, as with all new or complicated projects, the golden rule is to proceed in small, well-understood steps and to get each step working before moving on.

When we've finished, our Linux router will have three NICs with the following addresses:

Interface name	Description	Network	Address
eth0	External	192.168.0	192.168.0.3
eth1	Subnet 1	192.168.1	192.168.1.1
eth2	Subnet 2	192.168.2	192.168.2.1

Install the first network interface as you would if you were building a server from scratch, allowing Linux to detect it automatically. Linux will call the first interface eth0 by default and will usually pick up an IP address for it using DHCP, which works if your ADSL modem is set up to provide one. If not, you can assign the router's eth0 interface an address using whatever system configuration tool you have to hand, such as YaST. To do so manually, log in as root and type:

```
# ifconfig eth0 192.168.0.3 netmask
255.255.255.0 up
```

DHCP adds a default route automatically but, if your internet gateway doesn't do DHCP or you don't want to switch it on, you can add the default route yourself as follows:

```
# route add default gw 192.168.0.1 dev
eth0
```

As you can see in the diagram, 192.168.0.1 is the address assigned to the ADSL modem/router's internal interface. Open a web browser on the Linux router and enter www.google.com to make sure the router can connect to the web through the gateway.

Next, we need to add the second network interface to the router. We'll give this a static IP address of 192.168.1.1, as it will handle the 192.168.1.x network. In YaST and other graphical configuration tools, this is very easy. If you do not have access to YaST, then Webmin (www.webmin.com) or any other graphical configuration tools, you can set up the card using the ifconfig command as follows:

```
# ifconfig eth1 192.168.1.1 netmask
255.255.255.0 up
```

This is our first internal network interface. Connect a hub to it, and you can test the new interface by pinging it. This sends a succession of packets to the



Network Interfaces

Interfaces Active Now

Name	Type	IP Address	Netmask	Status
eth0	Ethernet	192.168.0.3	255.255.255.0	Up
eth1	Ethernet	192.168.1.1	255.255.255.0	Up
eth2	Ethernet	192.168.2.1	255.255.255.0	Up
lo	Loopback	127.0.0.1	255.0.0.0	Up

Interfaces Activated at Boot Time

Name	Type	IP Address	Netmask	Activate at boot?
eth0	Ethernet	From DHCP	Automatic	No
eth1	Ethernet	192.168.1.1	255.255.255.0	No
eth2	Ethernet	192.168.2.1	255.255.255.0	No
lo	Loopback	127.0.0.1	255.0.0.0	Yes

For those not blessed with a graphical configuration tool, web browser-based utilities such as Webmin can prevent the need to edit configuration files directly

YaST2

The routing can be set up in this dialogue. The **Default Gateway** matches every possible destination, but poorly. If any other entry exists that matches the required address, it will be used instead of the default route. The idea of the default route is simply to enable you to say "and everything else should go here".

Enable **IP Forwarding** if the system is a router.

Routing configuration

Default Gateway: 192.168.0.1

☐ Expert Configuration

Destination	Gateway	Netmask	Device

☒ Enable IP Forwarding

Remember to turn on IP forwarding in the kernel, or you'll have built a server instead of a router. This is simply a matter of clicking a box in YaST

interface, out to the hub and back again and is called a remote loopback test:

```
# ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84)
bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1
ttl=64 time=0.120 ms
```

Pressing Ctrl-C stops the ping command. Now you can install the third interface, giving it the IP address 192.168.2.1, and test its connection. Plug it into a hub so that the ping has a return path.

ROUTE FINDER

We now have a Linux box with three network interface cards. DHCP assigns an address to eth0, unless we've opted to give it a static one. We've assigned a different IP address to each of the two other cards. If we now plug a computer into one of the internal hubs and try to access the internet, it won't work. The packets will get as far as our new router, but they won't be sent on towards the ADSL modem and out to the internet.

For this to work we need to turn on a function in the router's Linux kernel called IP forwarding. This will forward the packets for us, according to the router's internal routing table. This is easy to turn on in YaST, as it appears as a tick box that affects all network interfaces (see the picture, top-right). To complete this step manually you should alter the content of the file `/proc/sys/net/ipv4/ip_forward`, setting it to 1, as opposed to the default 0. Type the following at the command line:

```
# echo 1 > /proc/sys/net/ipv4/ip_
forward
```

and reboot the router. Now we need to find out how the kernel plans to forward the non-local traffic that should go to the internet. Typing the route command shows us the routing table:

```
# route
Kernel IP routing table
```

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
192.168.2.0	*	255.255.255.0	U	0	0	0	eth2
192.168.1.0	*	255.255.255.0	U	0	0	0	eth1
192.168.0.0	*	255.255.255.0	U	0	0	0	eth0
link-local	*	255.255.0.0	U	0	0	0	eth0
loopback	*	255.0.0.0	U	0	0	0	lo
default	192.168.0.1	0.0.0.0	UG	0	0	0	eth0

As mentioned earlier, each of the three IP address ranges has an associated interface, such as eth0, eth1 or eth2. Whenever the router receives a packet destined for an IP address in a range such as 192.168.2.x, it looks up the relevant routing table entry and simply re-transmits the packet on the corresponding interface. In this example, data sent to a computer with the address 192.168.2.5 would be transmitted on the interface called eth2.

Packets that are not destined for the local networks are handled by the last entry in the routing table, called default. This specifies the default route. The kernel knows that if the routing table doesn't have a route that deals explicitly with a destination, it should pass the packets to the computer indicated by the default route. In this case, the router picked up the route automatically when eth0 obtained its IP address by DHCP from the internet gateway. The default route points to the internet gateway itself, so any packets that are not destined for either of the local networks are sent to the internet through the gateway.

HOUSE RULES

You might prefer to connect your new router to the internet using a standard ADSL modem, rather than connecting to an ADSL modem/router combination. If you do this and dispense with a firewalled internet gateway, you should activate the router's firewall to keep your network safe. We covered this in detail in *Linux Expert*, *Shopper* August 2004, but there are a few additional settings we must make so our router will route properly once secure.

We need to turn on forwarding and address masquerading. Because our ISP gives us only a single public IP address (using DHCP), and as we may have several computers on the internal networks, we need to fool the ISP's routers into believing that the router/firewall is the only PC using the connection.

Address masquerading achieves this by keeping track of connections to the internet initiated by the internal computers. Imagine that a Windows PC

with an IP address of 192.168.1.2 is trying to access www.google.com, which has the address 216.239.59.99. It sends a request to Google's website. The Linux router/firewall sees that the request has come from 192.168.1.2 and is destined for a network not known to its routing tables. It forwards it to the ISP, but not before changing the request subtly and replacing the Windows PC's IP address in the request with its own. The ISP believes that the firewall itself, which it recognises as a valid system, has asked for the Google web page.

The ISP forwards the request through the routers that make up the internet until it arrives at 216.239.59.99. Google replies, sending back the web page and the internet routes this answer all the way back to our firewall, through the ISP. The firewall finds out where to send this data by checking its records to see which system on our network asked for it. Address masquerading keeps track of the connections, so the firewall can see that 192.168.1.2 asked for the page. It delivers the data, and Internet Explorer on the Windows PC shows Google's home page. This technique is also known as Network Address Translation (NAT).

We can activate this feature, telling the firewall that it's running on a router, using the firewall configuration files or YaST. Here we'll look at the changes YaST makes to SuSE 9.2's `/etc/sysconfig/SuSEfirewall2` configuration file. Other distributions' firewall configuration files should look the same:

```
FW_ROUTE="yes"
```

We now need to turn on masquerading and tell the firewall which interface it should use to perform this function for us. The following lines in the configuration file do this:

```
FW_MASQUERADE="yes"
FW_MASQ_DEV="$FW_DEV_EXT"
```

We'll set the value of the variable `$FW_DEV_EXT` in the next step by telling the firewall which are the internal interfaces and which is the external one. We specify the external interface – the one that should do the masquerading – as follows:

```
FW_DEV_EXT="eth-id-00:40:4f:4b:9d:
ef".
```

The colon-separated value is the MAC address of the card to which Linux has assigned the name



eth0. You can find out the MAC address using the ifconfig command. Type it and look for the text that starts "HWaddr". These numbers should be prefixed with eth-id- to create the correct value for the above setting.

Similarly, we set FW_DEV_INT to specify the addresses of the internal interfaces. Separate them within the string using a single space:

```
FW_DEV_INT="eth-id-00:06:4f:0b:d3:a9
eth-id-00:50:ba:82:48:fd"
```

BETWEEN THE DEVIL AND THE DMZ

Last month we talked about setting up a demilitarised zone (DMZ) to protect our local network from our own public-facing systems such as mail and web servers. The idea is that if a hacker were to attack one of our servers successfully, he wouldn't be able to use his position on our network to attack the internal computers.

We can use our router in conjunction with the firewall to create a DMZ using one of our internal interfaces. The following configuration entry will do the job:

```
FW_DEV_DMZ="eth-id-00:50:ba:82:48:fd"
```

If you do this, remember to remove the MAC address in question from the definition of FW_DEV_INT, above. This tells the firewall that the network connected to this network interface is now a DMZ. The firewall knows that the PCs in the DMZ must not access the internal network. The internal computers, however, can access the PCs in the DMZ. This means they can still send and download email to and from a mail server in the DMZ, but if a hacker compromises it he will find a firewall blocking his access to the internal network.

HELLO WORLD

With all these settings in place, we can connect a client computer to see if the router allows us to see the outside world properly. Let's try this for the 196.168.1.0 network. Set up a client PC with the network configuration:

Parameter	Value
IP address	192.168.1.2
Subnet mask	255.255.255.0
Default gateway	192.168.1.1
DNS server	192.168.0.1

The internet gateway we're using in this example is an ADSL modem/router that runs DHCP and pretends to be a domain name server (DNS). What it actually does is forward DNS requests to the ISP.

Instead of assigning static addresses to each computer on the internal network, we could install a DHCP server on the Linux router and set it to hand out IP addresses and other settings to computers on one or more of our internal networks. To do this, set up the DHCP server on your router to work with both your internal interfaces, and configure it to assign addresses in the range of 2-254. Addresses 0 and 255 are always reserved, and we've already used 1 for the router. The following example shows the lines that should go into /etc/dhcpd.conf file to set up DHCP on the network 192.168.1.x. Note the line that starts with 'range', where we specify

TRAFFIC REPORTS

Now the router is up and running, it's useful to be able to monitor what it's doing. To do this we can use the Simple Network Management Protocol (SNMP). SNMP runs on port 161 using User Datagram Protocol (UDP), as opposed to the Transmission Control Protocol (TCP) that most internet services use. UDP doesn't guarantee packet delivery, but it works well enough. Almost too well, in fact, as it spews out the server's most intimate details in plain text. You can see how much by installing the SNMP package on the router itself, starting the service and running the snmpwalk command:

```
linux:~ # snmpwalk -v 1 -c public
localhost | more
SNMPv2-MIB::sysDescr.0 = STRING:
Linux linux 2.6.8-24-default #1 Wed
Oct 6 09:16:23 UTC 2004 i686
SNMPv2-MIB::sysObjectID.0 = OID: NET-
SNMP-MIB::netSnmpAgentOIDs.10
SNMPv2-MIB::sysUpTime.0 = Timeticks:
(4832765) 13:
SNMPv2-MIB::sysContact.0 = STRING:
Sysadmin (root@localhost)
SNMPv2-MIB::sysName.0 = STRING: linux
SNMPv2-MIB::sysLocation.0 = STRING:
Server Room
-- more --
```

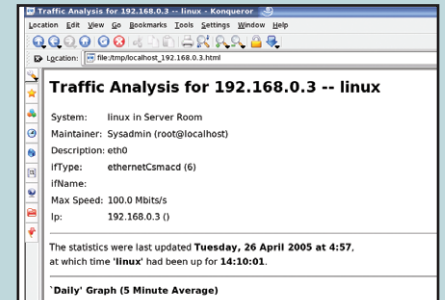
And so on, for page after page. SNMP collects data into a large tree structure, every node of which stores a value or a string. Snmpwalk 'steps' through this tree, reading and reporting all the values. You can also set some values; SNMP is not something you should expose to the internet as its security is very weak.

Most Linux distributions ship with a handy open-source utility called the Multi-Router Traffic Grapher (MRTG), which collects SNMP data and creates web pages containing graphs of information. This can be very useful for monitoring such things as hard disk and processor use or network traffic levels. If your distribution does not carry MRTG, you can download it free from <http://people.ee.ethz.ch/~oetiker/webtools/mrtg>.

The MRTG documentation is a good place to get an in-depth appreciation of its power, but we can get it up and running quickly using its cfgmaker script. The following command will create a skeleton configuration file that monitors traffic on the router's network interfaces:

the addresses that will be allocated to the client computers:

```
option domain-name "linux";
option domain-name-servers
192.168.0.1;
default-lease-time 14400;
ddns-update-style none;
subnet 192.168.1.0 netmask
255.255.255.0 {
    range 192.168.1.2 192.168.1.254;
    default-lease-time 14400;
    max-lease-time 172800;
}
```



Programs such as MRTG can display detailed information from the SNMP service for a huge range of system parameters, not just network traffic

```
# cfgmaker --output=/etc/mrtg/
mrtg.cfg --global "workdir: /tmp/
mrtg" ifref=ip --global 'options[_]:
growright,bits' public@localhost
```

We can now run MRTG with the subsequent configuration file to collect current data:

```
# env LANG=C /usr/bin/mrtg /etc/mrtg/
mrtg.cfg
```

Thanks to a bug in MRTG, you must run this command three times before it will complete without an error. This is simply because the program needs to move old versions of its output files, but it doesn't check first to see if there are any to move. To see its output, we use the indexmaker command:

```
# indexmaker --output=/tmp/mrtg/
index.html /etc/mrtg/mrtg.cfg
```

This command writes its output to a set of HTML files in /tmp/mrtg/. We can browse them by running a web browser on the same system to read the file. In the screen above we're using the Konqueror browser, which comes with SuSE's default desktop environment, KDE:

```
# konqueror /tmp/mrtg/index.html
```

You can click on the graphs to display more detail. To make MRTG run regularly, you could set up a cron job to run it and indexmaker regularly. MRTG and SNMP make a powerful combination. An excellent tutorial showing how to add other measurements is available at www.siliconvalleyccie.com/linux-hn/mrtg-advanced.htm.

Using Linux to build a router is more work than buying one, but if you follow our article all the way through you should have one up and running in an hour or two. You may also find that you gain a better understanding of how data moves through networks, and this will help when you are troubleshooting in the future. ☑

NEXT MONTH

BUILD YOUR OWN SECURE WEB SERVER
We show you how to get SSL encryption working with Apache